

# EIE522 Pattern Recognition: Theory and Applications

## Laboratory 3: Neural Networks for Pattern Recognition

### Objectives

1. To understand the working principles of a multi-layer perceptron and a convolutional neural network.
2. To get familiar with the error back propagation and stochastic gradient descent algorithms to train neural networks.
3. To use neural networks to solve challenging visual recognition problems.

### 1 Introduction

In this section, we first introduce two types of widely used neural network models, i.e. multi-layer perceptron (MLP) and Softmax classifier, and then we describe the convolutional neural network (CNN), which has attracted growing attention from the computer vision and machine learning community recently.

#### 1.1 Multi-Layer Perceptron (MLP)

MLP consists of one input layer, one or several hidden layers and one output layer, as shown in Fig. 1. The nodes between two neighbor layers are full connected. For each node in each hidden layer and the output layer, an activation function is applied:

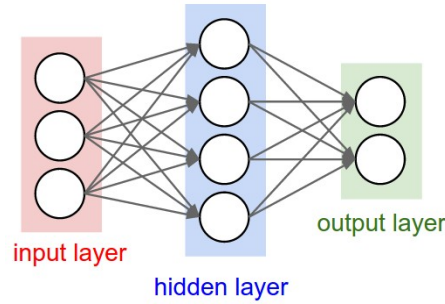


Fig.1. The structure of MLP.

$$\mathbf{a}^{l+1} = h(\mathbf{z}^{l+1}), \text{ with } \mathbf{z}^{l+1} = \mathbf{w}^{l+1}\mathbf{a}^l + \mathbf{b}^{l+1}$$

where  $\mathbf{a}^{l+1}$  denotes the output of the  $l + 1$  layer,  $\mathbf{a}^l$  indicates the output of the  $l$  layer,  $\mathbf{a}^0$  is the input,  $\mathbf{w}^{l+1}$  is the connection weights,  $\mathbf{b}^{l+1}$  is the bias,  $h$  is an activation function (e.g. the sigmoid function:  $h(x) = 1/(1 + e^{-x})$ ).

Suppose that we have  $N$  training samples, each training sample is categorized into one of  $K$  classes, then the ground truth label  $\mathbf{t}$  is a  $K$ -length vector with 0 and 1. For each training sample  $i$ , we can obtain the following error:

$$e_i = \frac{1}{2} \sum_{j=1}^K (a_j^i - r_j^i)^2 = \frac{1}{2} \|\mathbf{a}^i - \mathbf{r}^i\|_2^2$$

where  $e_i$  is the error,  $a_j^i$  is the  $j$ -th predict result, and  $r_j^i$  is the  $j$ -th ground truth of the  $i$ -th sample. We can further define the accumulated error of  $N$  training samples:

$$E = \frac{1}{N} \sum_{i=1}^N e_i = \frac{1}{2N} \sum_{i=1}^N \|\mathbf{a}^i - \mathbf{r}^i\|_2^2$$

If we consider the regularization term, the  $E$  can be rewritten as:

$$E = \frac{1}{2N} \sum_{i=1}^N \|\mathbf{a}^i - \mathbf{r}^i\|_2^2 + \frac{1}{2} \lambda \|\mathbf{w}\|_2^2$$

where  $\lambda$  indicates the regularization coefficient. The purpose of the regularization term is to make the small weights even small and eventually these weights will be close to zero (disconnected).

In general, back propagation with stochastic gradient descent is widely used to train the neural networks. We have:

$$\boldsymbol{\delta}^l = (\mathbf{w}^{l+1})^T \boldsymbol{\delta}^{l+1} \circ h'(\mathbf{z}^l)$$

where  $\boldsymbol{\delta}$  is the “errors” back propagated from output layer, the operator “ $\circ$ ” denotes the element-wise multiplication. For the output layer, the  $\boldsymbol{\delta}$  takes a slight different form:

$$\boldsymbol{\delta}^L = (\mathbf{a} - \mathbf{r}) \circ h'(\mathbf{z}^L)$$

where  $h(\cdot)$  is usually the sigmoid function. We can easily get the derivative of the sigmoid function as:  $h'(x) = h(x)(1 - h(x))$ . Finally, we have

$$\frac{\partial E}{\partial \mathbf{w}^l} = \mathbf{a}^{l-1} (\boldsymbol{\delta}^l)^T, \quad \Delta \mathbf{w}^l = -\eta \frac{\partial E}{\partial \mathbf{w}^l}, \quad \mathbf{w}^l = \mathbf{w}^l + \Delta \mathbf{w}^l$$

$$\frac{\partial E}{\partial \mathbf{b}^l} = \frac{\partial E}{\partial \mathbf{a}^l} \frac{\partial \mathbf{a}^l}{\partial \mathbf{b}^l} = \boldsymbol{\delta}^l, \quad \Delta \mathbf{b}^l = -\eta \frac{\partial E}{\partial \mathbf{b}^l}, \quad \mathbf{b}^l = \mathbf{b}^l + \Delta \mathbf{b}^l$$

## 1.2 Softmax Classifier

A Softmax classifier looks like the MLP with the difference lying in the activation function and the loss function used. For each hidden layer, the activation function used is the Rectified Linear Unity (ReLU):  $h(x) = \max(0, x)$ . For each training sample  $i$ , the cross entropy loss is defined as

$$L_i = -\log\left(\frac{e^{a_{t_i}}}{\sum_j e^{a_j}}\right)$$

where  $a_j$  is the  $j$ -th element of the class scores,  $t_i$  indicates the true class of the  $i$ -th training sample. The full loss of the training set is the mean of all the training samples:

$$L = \frac{1}{N} \sum_{i=1}^N L_i = -\frac{1}{N} \sum_{i=1}^N \log\left(\frac{e^{a_{t_i}}}{\sum_j e^{a_j}}\right)$$

Just like MLP, back propagation with gradient descent can be used to train the Softmax classifier. We define  $p_{t_i}$  as the probability of a training sample belonging to the  $t_i$ -th class,

then we have:

$$p_{t_i} = \frac{e^{a_{t_i}}}{\sum_j e^{a_j}}$$

The derivative of  $p_{t_i}$  with respect to  $a_j$  is:

$$\frac{\partial p_{t_i}}{\partial a_j} = \begin{cases} p_{t_i}(1 - p_j) & t_i = j \\ -p_{t_i}p_j & t_i \neq j \end{cases}$$

Then the derivative of  $L_i$  with respect to  $a_j$  is

$$\frac{\partial L_i}{\partial a} = -\frac{1}{p_{t_i}} \frac{\partial p_{t_i}}{\partial a_j} = (p_j - 1\{t_i = j\})$$

where  $1\{t_i = j\}$  is equal to 1 when  $t_i = j$  and otherwise is 0. We can further apply the chain rule to compute the derivatives of  $L_i$  with respect to  $\mathbf{w}$  and  $\mathbf{b}$  as the MLP classifier.

### 1.3 Convolutional Neural Network

Convolutional neural network (CNN) is a kind of deep learning model. Fig. 2 shows a simple CNN used in our experiment. It consists of two convolutional layers with each convolutional layer followed by a pooling layer. The convolutional layers extract the features from the input by applying a number of trainable filters or kernels sliding across the input image.

Each convolutional layer is followed by a pooling layer, which is used to reduce the spatial size of representation and alleviate the over fitting. The pooling layer takes small square blocks ( $s \times s$ , generally  $s = 2$ ) from the convolutional layer and subsamples it to produce a single output from each block. The most common pooling form is average pooling or max pooling.

After several convolution and max pooling layers, the final output is a 1-by- $n$  vector with each node indicating one class. Each neuron of the output layer fully connects to the nodes from the previous layer. The output is an  $n$ -way softmax predicting the probability distribution over  $n$  different classes.

The error back-propagation strategy and gradient decent algorithm used in training an MLP is adopted for training a CNN. However, training a CNN is more difficult and complex than training an MLP. The convolutional layer, pooling layer and full-connected layer take different BP (back-propagation) forms. More details can be found on the following websites:

<http://deeplearning.net/tutorial/lenet.html#lenet>

<http://cs231n.github.io/convolutional-networks/>

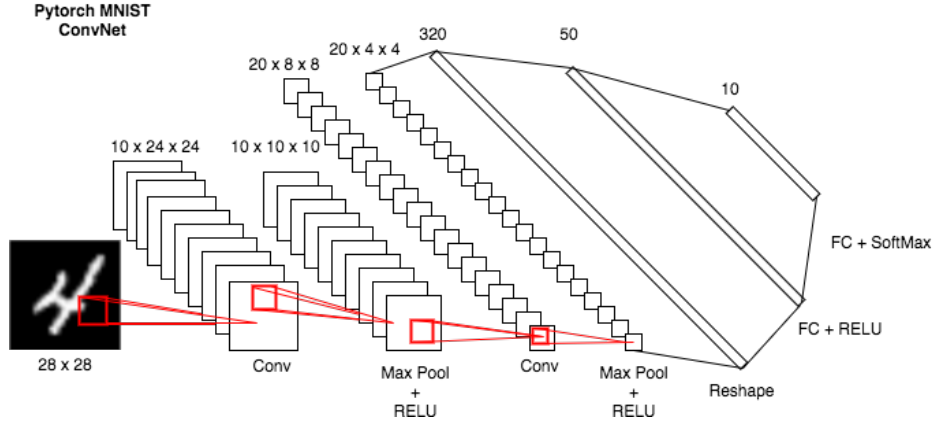


Fig.2. A simple CNN model used in our experiment.

### 1.4 Trainable Parameters

Although CNN model is a much deeper neural network than an MLP, sometimes the trainable parameters are not more than an MLP. Suppose we have an MLP with one hidden layer which contains 100 hidden nodes. The input size is 28-by-28, the output is a 1-by-10 vector, then we can compute the trainable parameters in the hidden layer as  $(28 \times 28) \times 100 + 100 = 78500$ , the number of trainable parameters in the output layer is  $100 \times 10 + 10 = 1010$ , the total trainable parameters for the MLP is  $78500 + 1010 = 79510$ .

For the CNN model, note that there are no trainable parameters in the pooling layer. Therefore, we only need to compute the parameters in the convolutional layers and the fully connected layer. We take the model shown in Fig. 2 as an example. In the first convolutional layer, the kernel size is 5, and the number of filters is 10. Therefore, the number of parameters is  $(5 \times 5) \times 10 + 10 = 260$ .

## 2 Data Sets

In this laboratory exercise, we will use neural networks to solve visual pattern recognition problems. MNIST is utilized in experiments. MNIST is a handwritten digit database. It has a training set of 60000 samples and a test set of 10000 samples, all the samples are categorized into one of ten digits: 0, 1, 2,...9. The digits have been size-normalized and centered in a fixed-size image. Fig. 3 shows examples of the MNIST databases.

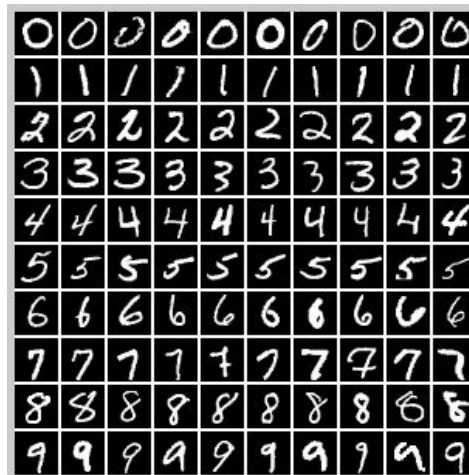


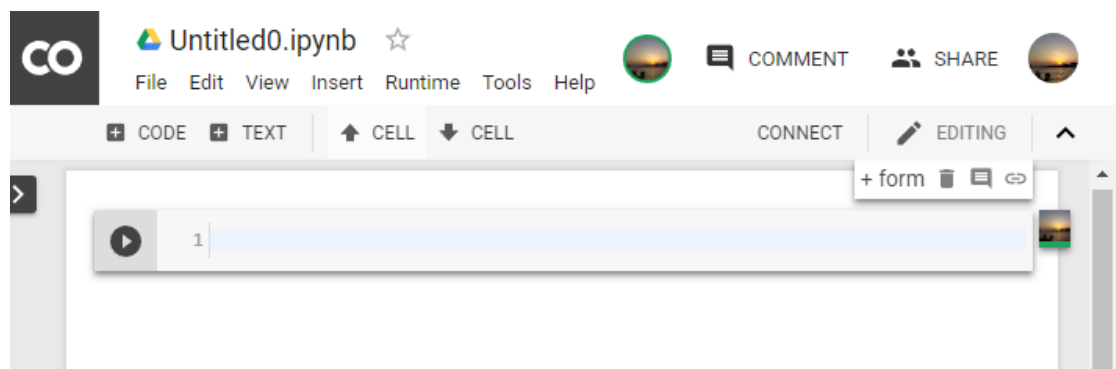
Fig. 3. Examples from MNIST database.



### 3 Google Colaboratory

In our experiment, we use the environment created by Google, called Google Colab. It is free of charge for you to run your program in a Nvidia Tesla K80 GPU, for at most 12 hours each time. In other words, the kernel will be shut-down every 12 hours, but you can still restart it immediately. In working space of Google Colab, you have to write the program in Python. In fact, Python supports most of the latest open-source deep learning libraries, such as Theano, Tensorflow, Keras, MXNet, and **PyTorch** (used in this laboratory), etc.

#### 3.1 Getting started

1. Go to <https://colab.research.google.com/?hl=en>. You will be redirected to a welcome notebook. (Optional) You can read it if you want to know more.
2. Sign in with you google account (top right corner). Create a google account if you do not have one.
3. Create a new **Python 3** notebook by selecting “New Notebook” → “New Python 3 notebook”. Or, you can create it by selecting “File” → “New Python 3 notebook”. The notebook will automatically be saved in your Google Drive account, under the “Colab Notebooks” folder.
4. Now, your Python notebook should be a blank page shown as follows.



5. You can add a code cell by clicking the “add code cell”  CODE button. Then, you can run the code inside a code cell by clicking the “run cell”  button.
6. Run the following code to print “Hello World”.



7. Use **GPU** by selecting “Edit” → “Notebook settings”. Change the hardware accelerator from “None” to “GPU” and then save it. Sometimes, an error will pop up, saying it can’t connect. This is due to most of the GPU machines being occupied. Therefore, you should re-try this step after a while.
8. Upload the Python notebook (*mnist.ipynb*) to Google Colab by selecting “File” → “Upload Notebook”.
9. Go through the notebook and run all the code cells, by selecting “Runtime” → “Run all”. You can always clear the outputs by selecting “Edit” → “Clear all outputs”, and restart the environment by selecting “Runtime” → “Restart runtime”.
10. Finally, you will need to modify some of the codes and answer the following questions.

**EIE522Pattern Recognition: Theory and Applications**  
**Laboratory 3: Neural Networks for Pattern Recognition**

Student Name:\_\_\_\_\_

Student Number:\_\_\_\_\_

1. What is the number of trainable parameters (i.e. weights and biases) in the 3-layer MLP and the CNN with default hyperparameters? Show your calculations.
2. Report the training and test accuracies of the MLP and CNN every 10 epochs. Discuss your observations.
3. Report the training and test accuracies every 10 epochs when layer-1 and layer-2 of the MPL are followed by ReLU activation? Discuss your observations.
4. Change the weight regularization coefficient (*weight\_decay\_coef*) from 0 to 0.00001 and to 0.01. Report the training and test accuracies of the MPL (with ReLU) and CNN every 10 epochs, and the final values (mean, min., max.) of the trainable parameters. Discuss your observations.